MINISIS and Recent Advances in the Theory of Data Base Systems

F.A. Daneliuk & W.S. Page*
Information Sciences Division
International Development Research Centre
Ottawa, Canada

Abstract

The paper describes an advanced data base management system, MINISIS, developed at the International Development Research Centre, Ottawa, Canada, from the point of view of recent developments in the theory of data base systems. The concepts of relations are introduced as a means of representing information in a computer system. Relations are then generalized and examined from the point of view of category theory. This enables us to describe the operations which MINISIS performs on relations in a mathematically rigorous way. The advantages of such a system are briefly examined.

Keywords: MINISIS, IDRC, data base theory, relational data base system, category theory

* Mr. Page was acting as consultant to IDRC while on leave from Statistics Canada, Ottawa, Ontario, Canada

1. Background to MINISIS

The MINISIS system was developed at the Information Sciences (IS) Division which is one of four program Divisions of the International Development Research Centre (IDRC), an organization which supports research to aid the process of development in the Third World. IDRC is funded by the Government of Canada, and has its head office in Ottawa, Canada. It also has regional offices in Singapore, Cairo, Bogota, Dakar, and Nairobi. The Centre has an international Board of Governors, including some from developing countries.

The Centre funds projects for both the gathering and dissemination of information with respect to development. Part of the mandate of IS is to support the development of tools to make existing knowledge available for the purposes of development. In its role of disseminating information, the Division is interested in computer systems which are available, and in ways of making them available, to developing countries. The paucity of commercial resources for processing bibliographic data and the expressed interest of several organizations (in both developed and developing countries) in acquiring such a facility on a low-cost computer, led the Division to initiate a research project which resulted in the development of MINISIS [4]. In January 1978, the first version (Version A) of MINISIS became operational in the IDRC Library. The present version of MINISIS (Version D, 1981) is licensed by IDRC and its agents to operate in over twenty centres world-wide.

2. MINISIS Data Base Management System

MINISIS is a data base management system which is readily usable as an information system because it is equipped with a set of seven generalized end-user processors. These processors make it possible for an unsophisticated user to collect and edit data in a data base, and then to manipulate it for purposes of retrieval (using logical operators), sequencing and printing reports, or simple computation. MINISIS also has a full set of processors for the data base manager which enable the simple definition and maintenance of data bases.

The design of MINISIS took cognizance of the fact that a generalized system should have three levels of operation - the external level which interfaces with the outside world, the internal level which interfaces with the machine, and the conceptual level which provides the interface between the external and the internal levels. This conforms roughly to the three schema approach of the ANSI/X3/SPARC study group on data base management systems [1].

The relational model of data was chosen for the conceptual level because of the great generality and simplicity of the model, as well as the availability of a mathematical theory of relational operations. We give an introduction to relational data base theory and an overview of recent advances later in this paper. In contrast to the conceptual level, neither the external nor the internal levels of MINISIS are purely relational. At the external level we must have the flexibility to receive data from and present data to the users in the format most convenient to them. For the reason of efficiency, at the internal level, we must conform as much as possible to the architecture of the operating system of the computer.

The conceptual level of MINISIS is the level at which the data base manager functions. The data base manager defines data bases in terms of relations and operations on relations, the operations resulting in additional relations. The collection of all definitions of relations for some application is referred to as a data model. Some of the relations exist as data stored in the mass storage of the computer, while other relations may exist only when they are referenced. The MINISIS system automatically performs all the operations necessary to create these virtual relations when required by an applications processor. In MINISIS, relations which have a stored representation (non-virtual) are called RD's. Virtual relations are of two types - PS's and DS's depending on the types of relational operations needed to create them. Any relation, virtual or not, can be thought of as a table. Each column of the table corresponds to an attribute (field) of the relation and each row to a tuple (record). Inside the table are the values of each attribute for each tuple.

An RD corresponds to a physical file on some external storage device. It is closest to the lowest level - the internal - in MINISIS. A data base defined as a RD always exists. The attributes of the RD contain values obtained from specified domains. The attributes of a relation may be revised over time, and the domains themselves may take on different characteristics. Some of the characteristics a domain may have are: (i)it may be tuple-valued; (ii)it may be set-valued; (iii)it may have fast access; (iv)it may be numeric; (v)it may contain only a fixed set of values, etc.

A MINISIS RD may contain any number of tuples (in the order of 10^7) and the tuples may contain any subset (including the full set) of the set of valid attributes defined on that relation. In other words, tuples may have undefined values for some of their attributes. Undefined values occupy no storage.

The next structure supported by MINISIS is the projected subset (or PS). The projected subset is a logical entity, unlike the relation, and exists only at the time at which it is activated. A data base which is defined as a PS is a subset of an RD that satisfies three conditions:

a) the projection must be defined on an existing RD;
b) the set of attributes which is defined for the PS must be a subset of those defined for the supporting RD; and
c) the set of tuples accessed through the PS must be a subset of the tuples contained in the RD.

Point a) is self-evident. Point b) represents the algebraic operation of project being executed on a relation. As part of the project operation, a domain may temporarily change its non-intrinsic characteristics; for example, its internal identifier may change, its fast access characteristics may change, and it may lose its set-value characteristics (this is called flattening). The latter may lead to many PS tuples where the RD had only one tuple. This does not really contradict point c) as it may seem. We will have more to say about flattening later in this paper. Point c) represents the algebraic operation of select (or restrict). Those tuples which are part of the PS must meet certain criteria - criteria based on some value of an

attribute. The selection is stated as a combination of Boolean and comparison operations on the contents of certain domains - or on the key of the underlying relation itself.

The most complex structure supported by MINISIS is what we have called the data submodel (or DS). The data submodel is a logical entity - a data base which exists only when it is activated. A DS represents the algebraic operation of join , in addition to select and project. Any set of RD's and/or projected subsets, within the same data model, may be joined together, to create another view of data. Those components which are projected subsets have, of course, all the attributes accruing to that structure. In addition, the DS as a whole may have the selection operation applied to it, and may have a project list defined for it, with the attendant possibility of temporarily changing the characteristics of the domains in the projection. A domain may become set-valued as a result of a join.

The join operation matches tuples from the components on specified attributes and merges them to form a simple tuple for the DS. This matching and merging may be done in a variety of ways. The conventional theory of relational joins defines only one such operation - the natural (also called intersection) join. T. Merrett of McGill University has defined several additional join operations [8] which result when tuples are allowed to have undefined values, specifically, the left , right and union joins. MINISIS supports all of these and, as we shall see in the next section, several additional join possibilities.

3. Introduction to Relations

To begin from the most basic concepts, consider the following sentences:

1) The IDRC central library has a monograph with the title " Database Systems " written by C.J. Date in 1977 , published by Addison-Wesley.

2) Our account with Prentice Hall of 123 Street, New York has a balance of 1000 dollars and the last shipment from them was on January 20, 1981.

3) The IDRC library in Bogota has a document series with the title " Small farming communities " that is authored by the Nairobi Centre of Agricultural Improvement in 1972.

We could of course extend this to a very long list if we wished to describe a real life situation. Everyone would agree that the sentences contain information (or data) and it is plausible to wish to store and manipulate the information in a computer system. We could choose simply to store the sentences in the textual form as given. We would soon discover, however, that this unstructured form has many disadvantages. An alternative is to group together sentences that have a similar structure. For example, sentences 1 and 3 above are obviously similar and different from sentence 2. Then in each sentence in a group we isolate the data (words) from those that simply make for easy reading (noise words). We have underlined the selected data in the sample sentences. Each sentence in a group has essentially the same noise words but may have different data. We could construct a skeleton sentence for each group with dummy data in place of the actual data. For example, a skeleton sentence for sentences 1 and 3 might be:

4) The <location> has a <type> with title <title> written by <personal author> of the organization <corporate author> in the year <date> and published by <publisher>.

A skeleton for sentence 2 might be:

5) Our account with <name> of <address> has a balance of <balance> and the last shipment was on <date>.

The dummy data items are enclosed in <>. One could think of them as variables or place holders. We will say that each such skeleton defines a relation. The dummy data items will be called attributes of the corresponding relation.

Now we might think of organizing the information contained in the sentences in the form of tables. We will have one table for each sentence type, i.e. skeleton or relation. The columns of a table are labelled with the attributes of a relation, while each row of a table contains the corresponding data from a simple sentence. Such rows are called tuples of the relation. We will also give each relation a name. Thus for the above example we construct two relations:

HOLDINGS (location, type, title, pauthor, cauthor, date, publisher)

VENDOR (name, address, balance, date)

ommitting the <> and abbreviating a little. The name of the relation is written outside the paranthesis and the attributes inside.

In table form we have:

HOLDINGS

Location	Type	Title	P-auth	C-auth	Date
IDRC Library Ottawa	Monograph	Database System	C.J. Date		1977
IDRC Library Nairobi	Series	Small Farming		Nairobi Centre	1972

Fig. 1. Relation as a table

This form of the information, while perhaps not very easy to read, is very suitable for storing in a computer system and in fact has an easily defined mathematical structure on which we will concentrate in the following discussion.

The skeletons above are equivalent to the concept of a predicate in symbolic logic. The dummy data items act as variables. In terms of logic a fully instantiated predicate corresponds to a sentence i.e. a skeleton with all the dummy data replaced with real data. A fully instantiated predicate is either true or false. We considered only true sentences above, i.e., facts. We could have also considered false sentences but we are not usually as concerned about storing and retrieving un-truths or non-facts.

A predicate or skeleton is defined by its inherent meaning and context, i.e. the manner in which we produced a skeleton in our examples. In what follows we will not be concerned with representing the meaning of predicates mathematically though this has been done by logicians. Instead we will discuss models of predicates. A model of a predicate is just a relation.

4. Conventional Mathematics of Relations

A relation has been defined [3] as a subset of the cross-product of a collection of sets, in symbols,

$$R \subseteq D1 \times D2 \times D3 \times \ldots Dn$$

A set is just a collection of unique values. We often write the collection inside braces, e.g. {1,2,3} is a set containing three elements - three unique values. Each of the Di above is a set containing some values. For example, D1 might be a list of names, D2 might be the set of all positive numbers, D3 might be a set of dates etc. We form the cross-product, denoted by x, by creating a new set with elements that are groups of values called tuples. Each tuple contains a single element from each underlying set. A subset is a part osf some other set. Thus R, a relation, is a set of tuples selected from the cross-product.

We could say that the cross-product is the biggest possible relation.

The sets Di are called the domains of the relation.

In relating a relation to a predicate we say that an attribute (or predicate variable) takes on values from a domain of the relation. The subset of tuples included in the relation are those which make the predicate true. There is a rather subtle distinction here between attributes and domains that is often missed and has led to some confusion in the early literature on the subject of relations. In terms of relations, an attribute is defined positionally, i.e., the value of i in Di. Thus, a relation has a first, second, third, etc.,

attribute. However, two attributes may take values from the same domain, e.g., D1=D2; D1 and D2 are in fact the same set. Although we may sometimes use the term attribute and domain interchangeably it is best to keep the distinction in mind. We will use the notation:

R (A1:D1,A2:D2,...)

where Ai is the name of an attribute and Di is the name of a domain. We will require that all of the Ai are unique names while the Di may be identical. Thus we will not have to be concerned about position. For example the HOLDINGS relation would be written:

HOLDINGS (LOC:locations, TYPE:types, title:string, p-author:name, c-author:name, published:date, publisher:name)

Reducing general information to the form of relations considerably simplifies the problem of representing data in the computer. In the above examples an obvious solution to the storage of relations is to make each relation into a file. A file is a storage area in the mass storage of the computer system which consists of a number of individually accessible records. A deck of punch cards on older generation computer equipment qualifies as such a file - each card is a record. Each record of the file will contain a single tuple of the relation. Within each record separate areas or fields are assigned to each attribute of the relation.

Unfortunately not all relations can be represented in such a simple form. It is likely that one might wish to define a relation where one or more of the attributes for some tuple refers to a set of values. In such a case the corresponding domain must be a set of sets. In the example above it is feasible for a book to have more than one author.

To accommodate this in the file representation of a relation we would need to have more sophisticated ways of assigning space in a record to attributes. Again there are some obvious solutions. We might define special field separator codes to distinguish between one value of the field and the next. Unless we are willing to reserve a fixed amount of storage space for each such set of values, each record of the file will be of different length. Already the idea of a deck of cards seems inadequate. In order to deal with these and other problems of representation it is desirable to restrict the type of relations that can be represented. When we discuss operations which form new relations from existing relations, we will see that it is possible to restrict the type of relations stored without losing the ability to store the more general information. It is possible to de-compose complicated relations into simpler forms which meet the restrictions and later re-combine them when necessary.

One way to restrict the form of a relation is to demand that the domains of a relation be simple sets. In this case elements of domains are limited to just atomic values, e.g. a simple number, or a single string. Of course, the definition of atomic values is somewhat arbitrary and depends on the nature of the computer system and programming language chosen. A relation which has domains which are only simple sets is said to be in first normal form. We may represent the tuples of a relation in first normal form as simple "flat" records, i.e. records in which there is no additional structure beyond division into a field for each attribute.

A relation is said to have a candidate key if some subset of its attributes determine uniquely a single tuple of the relation. Since a relation, in the above definition, is a set and must therefore have unique elements it is always possible to take the subset as the whole set of attributes and obtain a trivial candidate key. The smallest subset of attributes which uniquely determine a tuple is usually called the key of the relation. The key of a relation is determined by the nature (meaning) of the associated predicate and not by the values of the tuples actually present at any one time in the relation. In our earlier examples it is clear that name is the key of VENDOR; however it is not so clear what the key of HOLDINGS might be. Keys are important for simplifying the process of locating a particular tuple or record and are the basis for the development of more advanced file systems, very much like the indexed sequential access method.

A relation is said to be in second normal form if no proper subset of attributes in the key of the relation determine any of the attributes not in the key. In other words, the key cannot be made smaller by ignoring some (but not all) of the non-key attributes of the relation.

The third normal form requires that the only relationships present in a relation be those that lead from the key of the relation to the other attributes and that there be no relationships between the non-key attributes. Again the definition depends on the meaning of the associated predicate. We can observe that VENDOR is certainly in third normal form.

Relations in third normal form have the property that deletions or additions of single tuples of the relation can be performed without inadvertently losing information or violating the meaning of the associated predicate.

A classic example of a relation in first and second normal form but not third normal form is given by the skeleton sentence or predicate:

1) Joe Smith's office number is B101 and telephone number is 123-4321.

2) The office number of <name> is <office> and his telephone number is <phone>.

3) DIRECTORY ((NAME:PERSON), OFFICE:CODE, PHONE:NUMBER)

For the purpose of the example we assume that all persons have unique names, thus NAME is a suitable key of the relation in 3) above. We indicate the key by an extra set of parenthesis around it, bearing in mind that a key may consist of more than one attribute. Each person is assigned a single office and a single telephone. Notice however that to each office we also implicitly assign a telephone number and therefore the relation is not in third normal form.

If DIRECTORY is the only source of information in which telephone numbers are assigned to offices then it is clear that deleting a tuple, i.e. vacating an office will cause an unintended loss of information. One solution is to break DIRECTORY into two small relations as follows:

DIR1 ((NAME:PERSON), OFFICE:CODE)

DIR2 ((ROOM:CODE), PHONE:NUMBER)

Now vacating an office involves deleting a single tuple of DIR1; DIR2 remains unchanged, i.e., we do not necessarily remove the telephone from the room. No information is lost. It is obvious that it is possible to re-create DIRECTORY when we need it by matching tuples from DIR1 with tuples from DIR2 where the value of the OFFICE attribute of DIR1 matches the ROOM attribute of DIR2. This is an example of the relational operation of join which we will now examine in more detail.

5. Relations in Category Theory

It will be convenient to develop a pictorial form to display the structure and interrelationship of relations. We will represent relations and domains (objects) as vertices in a labelled directed graph. Attributes of a relation are represented by edges (arrows) directed from the relation to a domain. The examples of DIR1 and DIR2 above are represented in the following diagram:

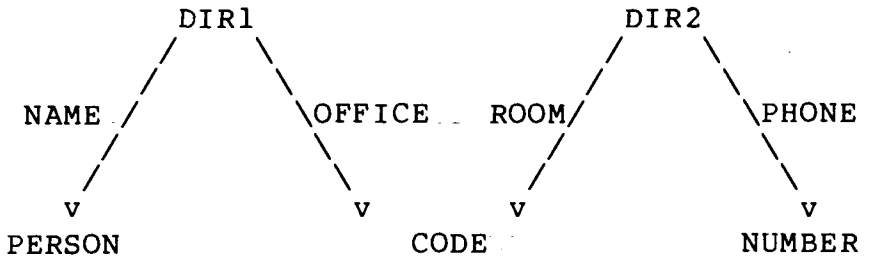

Fig. 2. Relations as diagrams

What we have drawn is actually a diagram taken from the mathematical category of SET. Each object in SET is literally a set, each arrow in SET is a function from one set into another. A function f from object A to object B (written f:A->B) is an operation which assigns a single element of the set B to each element of set A. While every element of A must be associated with some element of B, it is not necessary that all elements of B have some corresponding element of A. It is even possible that a single element of B corresponds to more than one element from A. The set A is called the domain of f. The set B is called the co-domain of f. A function is also called a many-to-one mapping from A to B. The recent developments in the

mathematics of TOPOS theory [6] has shown that it is possible to replace entirely the notion of sets and elements with the notions of objects and arrows. It will not be necessary to go into the details of TOPOS theory for the purposes of this paper. We will, however, use some of the elementary concepts of categories. Any collection of objects and arrows is a category provided that [2]:

i) for each object A there is a unique identity arrow id(A);

ii) for each pair of arrows (a:A->B, b:B->C) where the co-domain of a is the domain of b there is another arrow called the composition written b*a:A->C;

iii) the composition of an arrow b:A->B with the identity arrow of either the domain or co-domain objects is identical to b ; id(B)*b=b*id(A)=b;

iv) The composition of three arrows (a,b,c) defined by composing the first two and then composing the result with the third is identical to performing the composition in the other order; (a*b)*c=a*(b*c).

A TOPOS is a category with additional rules that we need not reproduce here.

In the category SET (in fact in any category that obeys the TOPOS axioms) for any pair of objects, say A and B, there exists an object denoted A x B and two arrows a:A x B -> A and b:A x B -> B such that for any other two arrows from some other object, a':C -> A and b':C -> B there is a unique arrow h:C->A x B with a'(c) = a(h(c)) and b'(c)=b(h(c)) for all elements c in C. The object A x B is just the familiar cross-product of sets A and B. The arrows a and b correspond to projection functions that map the tuples of A x B to their respective components, i.e. a(<x,y>)=x, b(<x,y>)=y where <x,y> denotes a tuple of A x B. Equations such as a'(c)=a(h(c)), for all c in C is usually written a'=a*h.

The above theorem, generalized to arbitrary finite cross-products, i.e., D1 x D2 x ... x Dn, allows us to represent a relation as a single arrow into the cross-product, rather than as a collection of arrows into the domains. This leads to a natural generalization of the idea that a relation is a subset of the cross-product. Subsets in TOPOS theory are represented by arrows with the property that corresponds to a function being a one-to-one mapping, i.e. an injective function. If the function h:C->A x B is injective then the subset of A x B identified by h is just those elements of A x B that have a corresponding element in C under the mapping. If h is allowed to be a general function then the result is a generalization of the concept of a relation. A generalized relation (also called a source) is not necessarily a set in exact terms. We no longer require that the tuples of the relation are all unique. Generalized relations may not be in second normal form even when all attributes are taken for the key.

In more concrete terms, if we take the example of the DIR1 relation above, we could implement the set DIR1 in a computer system as the set of addresses of records in some file. At each address in the file is stored a single record. Such an address is often called the internal sequence number (or ISN) of the record. Each record contains two fields corresponding to the attributes NAME and OFFICE, whose values come from the sets PERSON and CODE respectively.

We are now ready to describe in mathematical terms the concept of the join of two relations. Similar to the definition of the cross-product above, we have the following theorem:

For each pair of arrows a:A->B and c:C->B sharing a common co-domain object B, there exists another pair of arrows a':D->A and c':D->C, sharing a common domain object D with a*a'=c*c' for all elements in D such that any other pair of arrows a":E->A and c":E->C having a*a"=c*c" have a unique arrow h:E->D. Since a picture is worth a thousand words we can represent this as the following diagram:

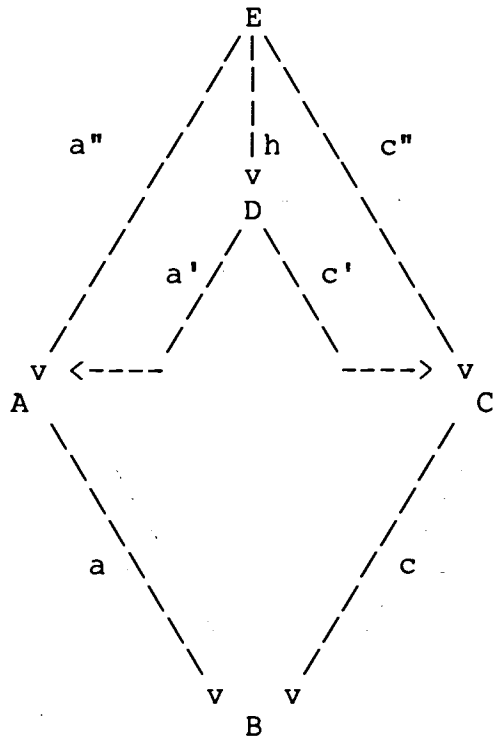

Fig. 3. The Pullback Diagram

To summarize, we have said that the pair of arrows (a,c) determine a unique pair (a',c') such that for any other pair (a",c") there is a unique arrow h, and further, that any two paths between two objects in

the above diagram are equivalent to each other. A diagram where any two paths between two objects are equivalent is called a commutative diagram. Commutative diagrams are the main tool of category theory for representing a complicated collection of relationships in a single form.

The pair of arrows (a',c') is called the pullback of the arrows (a,c) and the set of arrows (a',c',a,c) is called a pullback square. Pullbacks are a very important concept in TOPOS theory.

Now, we see that in Fig. 2 above we can form the pullback of arrows (OFFICE, ROOM) as in the diagram in Fig. 4.

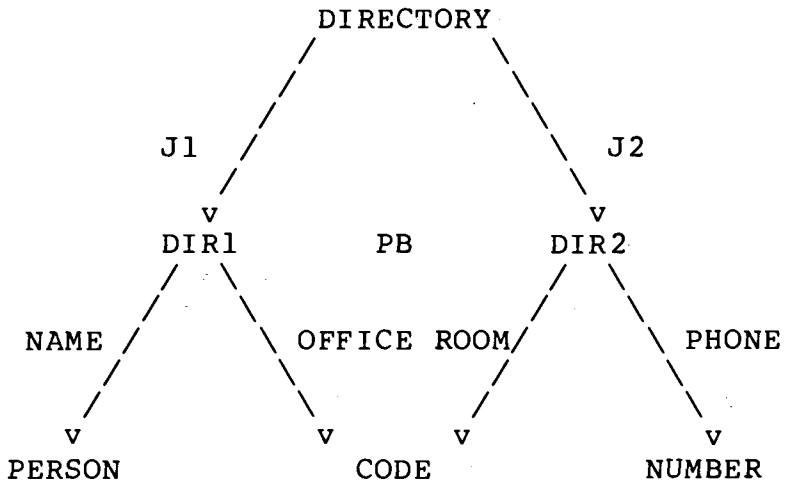

Fig. 4. Join of two relations

It turns out that DIRECTORY in Fig. 4 behaves exactly like the DIRECTORY relation of Fig. 2. We may think of DIRECTORY as a relation with two attributes, J1 and J2, with domains that are other relations. Alternatively we might think of DIRECTORY as a relation with three

attributes defined by composing arrows J1 and NAME, J2 and PHONE, and either J1 and OFFICE or J2 and ROOM. The last two compositions are equal because of the definition of the pullback.

It is outside the scope of this paper to explore further the formal properties of the pullback but we might note here a few of the more intuitive properties. In some (well definable) sense the pullback or join is the best possible representation of the relationship between DIR1 and DIR2 above. DIRECTORY does not, however, contain any more information than is present in the pair of arrows (OFFICE, ROOM). In the spirit of category theory we might say that the arrows (OFFICE, ROOM) together with the object CODE represent a co-relation. The operation of pullback can be seen as finding the best approximating relation for a given co-relation. It is interesting to ask the question of when this approximation is exact. It turns out this happens precisely when a relation obeys a type of restriction known as a multi-valued dependency.

6. How MINISIS Relates to Data Base Theory

We have seen above that relations, while very general are, in principle, often restricted to conform better to the capabilities of computer systems. Many data base systems based on relations require that all stored relations obey all three normal forms defined above. It has been argued that these restrictions place a heavy, though not impossible, burden on data base designers and even to an extent on the end-user. MINISIS, while making use of relational concepts, does not restrict stored relations according to the three normal forms. Even in MINISIS, however, it is necessary to place some limitations on the form of relations. According to the above diagrammatic conventions we might represent a typical MINISIS relation as follows:

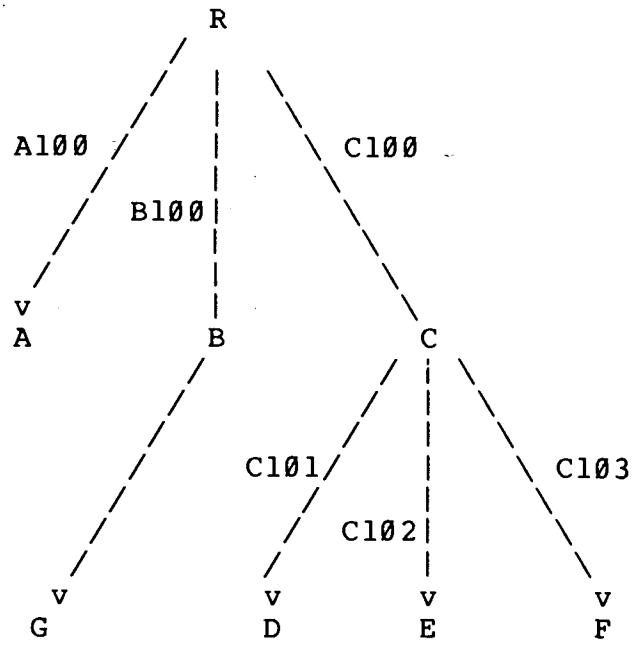

Fig. 5. A typical MINISIS Relation

In more conventional terms, the above example might be described as follows:

R is a file with three fields (A1Ø0, B1Ø0, C1Ø0).

The field A1Ø0 contains only single, simple values. Field B1Ø0 contains multiple (or none) simple values. Field C1Ø0 is a repeating group. Each group contains the sub-fields C1Ø1, C1Ø2 and C1Ø3. The sub-fields all contain single simple values.

Thus MINISIS allows relations to be simple first normal form relations or to be co-relations with co-domains that are either simple or are themselves first normal form relations. MINISIS relations can also be combinations of relations and co-relations of the above form. Repeating fields and repeating groups are implementations of the mathematical co-domain/co-relation concept referred to briefly above. One should be careful to note the reversal of terminology of domain and co-domain when speaking of arrows in a category versus relations. The domains of the relation are actually the co-domains of its outgoing attribute arrows, while the co-domains of a relation are the domains of its in-coming attribute arrows. We will call the above restrictions and extensions to the conventional concept of a relation the MINISIS Normal Form or MNF.

The directions of the arrows in Fig. 5 are very important to its meaning. It will be recalled that an arrow corresponds to a function or a many-to-one mapping, i.e., from "many" to "one" in the direction of the arrow. The basic idea, then, is that from the point of view of R, many file addresses or ISNs may have the same value of some simple

field, though each has only one value for that field, looking forward along an arrow. If we look backwards along an arrow, each ISN can have many occurrences of some multivalued field or repeating group. Each occurrence, however, belongs to only one ISN and has only one associated value. Notice that the same value may occur many times in a multi-value field or repeating group. In the above example A, G, D, E and F are the domains from which values are taken. B and C represent sets of "addresses" within each record.

MINISIS has facilities for transforming an MNF relation to another MNF relation and for joining two or more MNF relations to produce a new MNF relation. Because MNF is more general than the first normal form defined above, we must appeal to the generalized notion of a relation in the categorical sense rather than conventional set-theoretic notions in order to provide a complete mathematical theory of the MINISIS operation on relations.

The MINISIS RD, as mentioned above, is the stored form of an MNF relation. Each tuple of the MNF relation has a unique ISN. The ISN is mapped to the actual physical location in storage by an auxiliary file called the cross-reference file. The tuples are stored in a physically contiguous area of the mass storage in another file referred to as the master file. A tuple is physically represented by a variable length record with a directory portion and a data portion. The directory allocates space in the data area for each attribute value present in the tuple.

The MINISIS PS transforms an RD to another (virtual) MNF relation in three different ways. First a PS may form a new relation by simply selecting a subset of the attributes of the RD. We have a tuple in the PS MNF relation for each tuple in the RD. The resulting relation has the same address set (ISNs) as the RD. The generation is similar to that of the conventional relational projection, however, no attempt is made to eliminate duplicate tuples.

The second operation that can be performed by the PS is the selection of a subset of the address set of the RD as the address set of the resulting relation. Mathematically, this is represented by a one-to-one mapping from the address set of the PS into the address set of the RD, with the attributes of the PS defined via composition. In logical or semantic terms we could view this operation as adding extra clauses to the skeleton of the RD. In the VENDOR example above we might adjoin the phrase: "where balance is greater than 100 dollars.". In conventional relational terminology this operation is known as restriction or selection.

The third operation that can be performed by the PS is to reverse the direction of an incoming attribute arrow, i.e. to convert a co-domain to a domain. This has sometimes been called flattening in reference to conversion of un-normalized relations to first normal form. This operation can be understood mathematically in terms of the pullback concept defined above. Fig. 6 illustrates how the pullback defines a unique "best" relation representation of a co-relation.

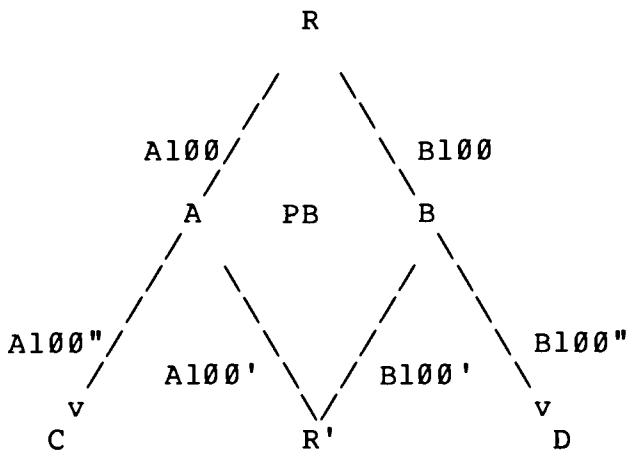

Fig. 6. Flattening as pullback

The MNF relation R(A100,B100) which is in fact a co-relation gives rise to a first normal form relation on domains C, D and R defined by the following compositions.

R'(A100*A100':R, A100"*A100':C, B100"*B100':D)

There is a tuple for each element of the address set R' for each combination of occurrences of co-domains A and B for each element of R (ISN). R' represents R except where a co-domain is empty.

The MINISIS PS flattening operation acts on a single co-domain to produce multiple tuples for a single RD tuple, each with a single, simple value of the specified co-domain. The problem of representing empty co-domains as domains is solved by allowing an attribute of a tuple to be undefined.

The MINISIS DS defines a new MNF relations as the join of a number of RD's and PS's on specified attributes. If the attributes chosen for the join are strictly domains of the relations, i.e. non-repeating simple fields, then the operation of the DS can be described

mathematically as we described the relational join operation above. More interesting situations arise when we consider joins on co-domains. Fig. 7 illustrates how the pullback again defines a unique "best" relation:

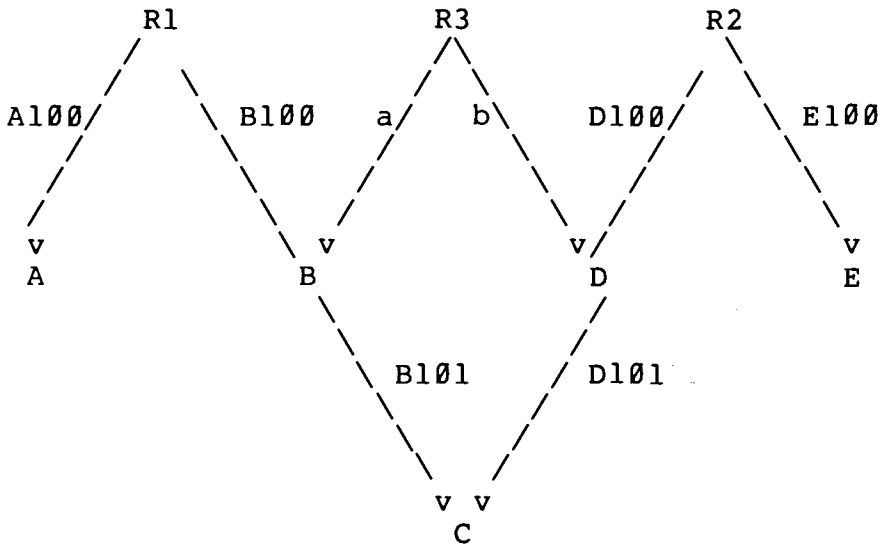

Fig. 7. Join on co-domains

The MNF-relations R1 and R2 are shown joined on attributes B100 and D100 respectively to produce the relation R3. The attributes of R3 are defined by compositions as usual. R3 has a single simple attribute with a value from the domain C corresponding to the value of the occurrences of the co-domain B and D. The remaining attributes are defined by "following" the arrows B100 and D100 from the specified occurrences of B and D. The result is reminiscent of "flattening" above and has been called "virtual flattening" in MINISIS terminology.

If we include the arrows from R3 to R1, R3 to C, R3 to E induced by the compositions and ignoring certain other arrows and objects, we arrive at the MNF relation displayed in Fig. 8.

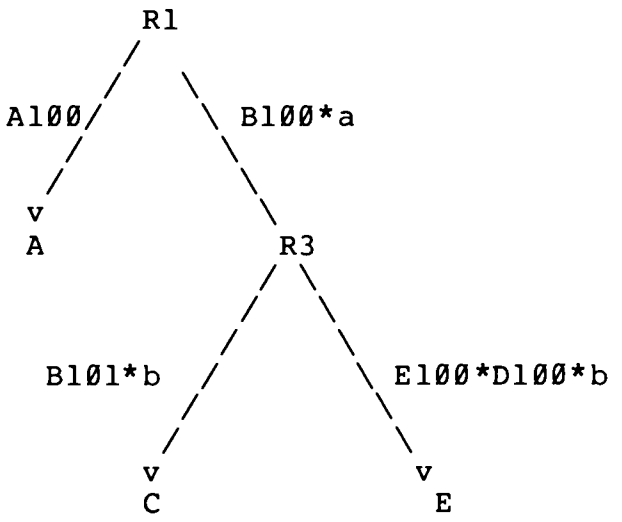

Fig. 8. An inner-left join.

We may view the relation of Fig. 8 above as a candidate for representing the result of the join shown in Fig. 7. Notice that the address set of the relation above is the same as one of the original relations. R3 has become a co-domain of R1, with subfields that contain information from the relation R2. It is natural to identify the co-domain R3 with the original co-domain B of R1. We may view the operation of the join as adding new subfields to the repeating group B. This form of join is referred to as an inner left join in MINISIS. Note that the relation R2 may not have co-domains other than D, otherwise the result is not representable in MINISIS normal form.

Finally we may note that the DS operation defines several additional join types related to the possibility of having undefined values of simple attributes. We make no attempt to define these operations mathematically in this paper. A rigorous definition of these joins has been made elsewhere [7,8].

7. Discussion

The MINISIS system represents the successful merging of theory and practice to produce an integrated, usable system. Rather than adopt an ad hoc approach in the development of this system, relational theory and the "three schemata" approach were successfully used as the design basis. This approach has led to a system which has a number of distinct advantages.

The most obvious advantage realized is the separation of functions through the three level approach. This separation makes it possible to focus on the needs of each level, and thus to tailor each level to its function. Thus, the external level is tailored to users who have no knowledge of programming; the internal level is tailored to the architecture of the physical computer itself; and the conceptual level is designed to be theoretically sound.

At the external (or end-user) level, the processors which implement the functions of data entry, data retrieval, etc., are completely generalized. In other words, they perform many functions on <u>any</u> data base - as opposed to specialized programs which perform particular functions on particular files. This generality makes it possible for users, who have no knowledge of computer programming, to develop sophisticated information systems on their own. For example, many MINISIS sites are run by individuals with no previous experience with computer systems.

The conceptual level enforces a standardization which would not otherwise be possible. It provides a fixed set of rules which define the manner in which data bases may be created and manipulated. These rules are derived from the theory of relations. New processors developed for the external level therefore are guarranteed to conform with the other processors. Furthermore, because MINISIS operations can be described accurately by a mathematical model, the behaviour of the system is predictable, even in circumstances unforeseen by the designers. The predictability, and thus the reliability, of the system, filters down into the internal level as well. The physical core is one which has guaranteed characteristics, with standard access paths which ensure the physical integrity of the data.

The development of computer systems to solve pressing real world problems has often gone ahead in spite of the absence of an adequate theoretical understanding of the problems. Thus we have seen the development of sophisticated file systems, sorting and merging packages, and even complete data base systems such as the hierarchical and network approaches, without a clear understanding of their limitations or their generality. The relational theory of data bases was the first attempt to develop a sound theoretical basis for data base systems design. In some ways the relational theory might be compared to the original primitive file systems developed in the 1950's and 1960's. We have seen in this paper that it is possible now to move beyond the original ideas of relational data base theory into the realm of category theory and topoi and remain on sound theoretical footings. We anticipate that the categorical approach to data bases will lead to further understanding of the issues of semantics in data base systems and to more advanced techniques of data base design.

Acknowledgements

The authors would like to gratefully acknowledge the many stimulating discussions on the subjects of this paper with T.H. Merrett of McGill University, Montreal and S. Bainbridge of the University of Ottawa, Ottawa, Canada.

References

[1] ANSI/X3/SPARC Study Group on Data Base Management Systems: Interim Report. Collection of ACM SIGMOD, 1975, vol. 7, no. 2.

[2] Arbib, M.A. and Manes, E.G. Arrows, Structures and Functions, Academic Press 1975.

[3] Codd, E.F. A relational model of data for large shared data banks. Communications of the ACM, 1970 vol. 13 no. 6 377-387.

[4] Daneliuk, F.A. The design and implementation of a data base system for bibliographic applications on a minicomputer. McGill University, Montreal, Technical Report SOCS-79.14, 1979.

[5] Date C.J. An introduction to database systems, 2nd edition - Reading, Mass. Addison-Wesley Publishing Co. 1977.

[6] Goldblatt, R. Topoi; the categorial analysis of logic. North Holland, 1979.

[7] Merrett, T.H. Relations as programming language elements. Information Processing Letters, 1977, vol. 6, no. 1, 29-33.

[8] Merrett, T.H. ALDAT - Augmenting the relational algebra for programmers. 1977. McGill University, Montreal, Technical Report SOCS-78.1.

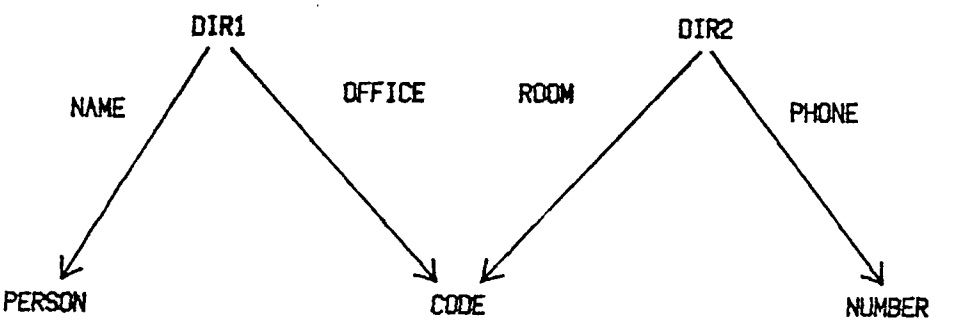

Fig 2. Relations as
diagrams.

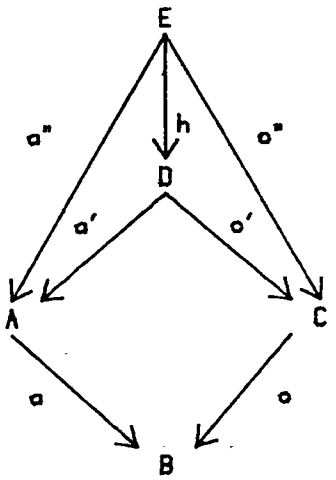

Fig 3. The Pullback diagram.

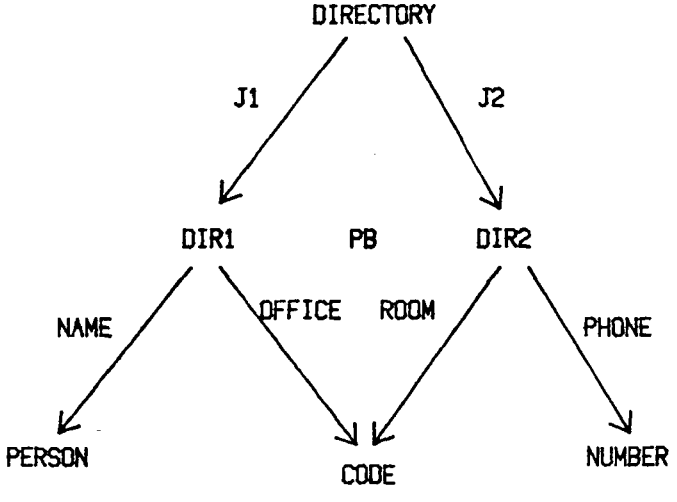

Fig 4. Join of two relations.

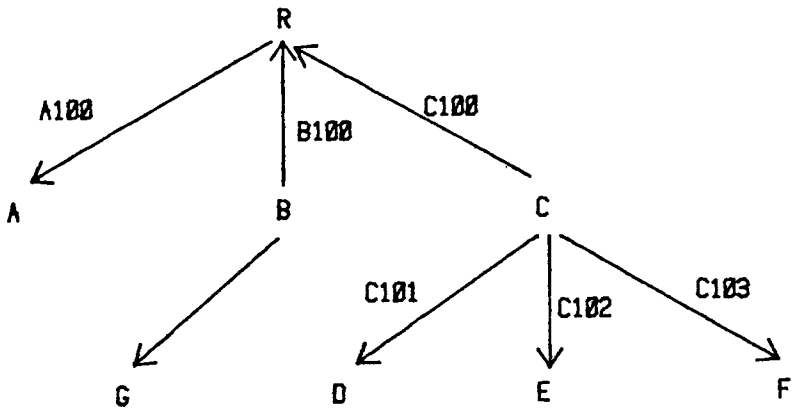

Fig 5. Typical MIN-ISIS relation

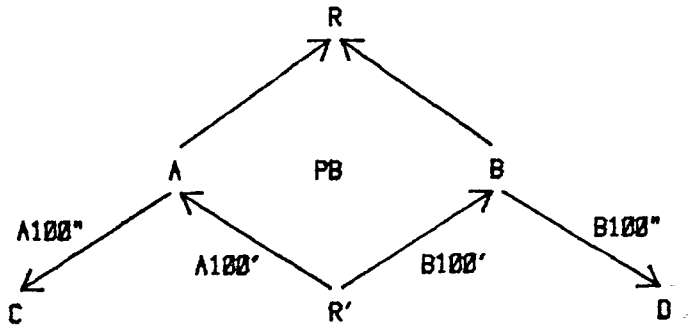

Fig 6. Flattening as
a pullback.

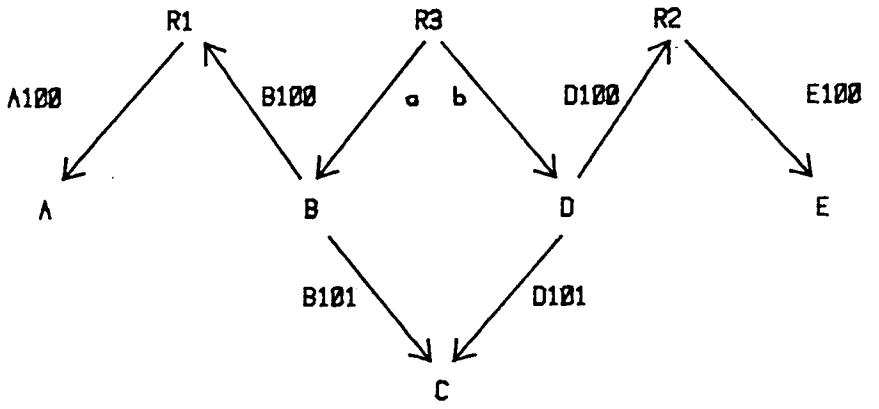

Fig 7. Join on co-domains.

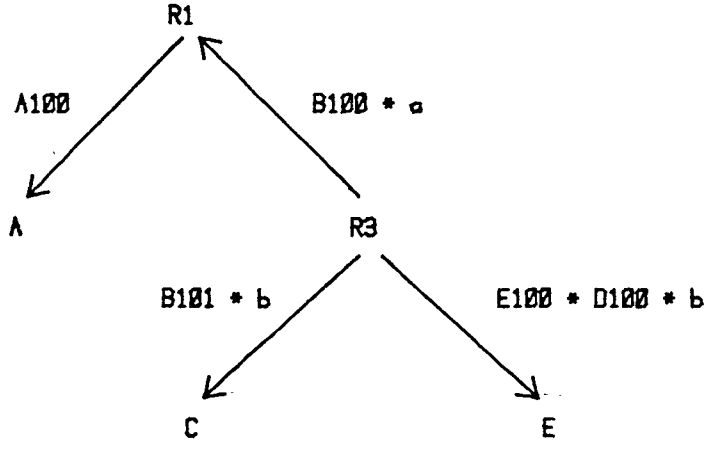

Fig 8. An inner-left join.